Symbolic Evaluation/Execution

Reading Assignment

* not required reading

Move from Dynamic Analysis to Static Analysis

• Dynamic analysis approaches are based on sampling the input space
  • Infer behavior or properties of a system from executing a sample of test cases
  • Functional (Black Box) and Structural (White Box) approaches

Static Analysis Approaches

• Dependence Analysis
• Symbolic Evaluation
• Formal Verification
• Data Flow Analysis
• Concurrency Analysis
  • Reachability analysis
  • Finite-state Verification
• Concurrency Analysis
  • Reachability analysis
  • Finite-state Verification
Today's Reading Material


* not required

Symbolic Evaluation/Execution

• Creates a functional representation of a path of an executable component
• For a path Pi
  - D[Pi] is the domain for path Pi
  - C[Pi] is the computation for path Pi

Functional Representation of an Executable Component

X_i is the domain of path P_i
Denoted D[P_i]

X = D[P_1] \cup \ldots \cup D[P_r] = D[P]

D[P_i] \cap D[P_j] = \emptyset, i \neq j

Representing Computation

• Symbolic names represent the input values
• The path value PV of a variable for a path describes the value of that variable in terms of those symbolic names
• The computation of the path C[P] is described by the path values of the outputs for the path

Representing Conditionals

• An interpreted branch condition or interpreted predicate is represented as an inequality or equality condition
• The path condition PC describes the domain of the path and is the conjunction of the interpreted branch conditions
• The domain of the path D[P] is the set of input values that satisfy the PC for the path

Example program

```
procedure Contrived is
    X, Y, Z : integer;
    1 read X, Y;
    2 if X \geq 3 then
      3 Z := X+Y;
    else
      4 Z := 0;
    endif;
    5 if Y > 0 then
      6 Y := Y + 5;
    endif;
    7 if X - Y < 0 then
      8 write Z;
    else
      9 write Y;
    endif;
end Contrived;
```

<table>
<thead>
<tr>
<th>Stmt</th>
<th>PV</th>
<th>PC</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>X\leftarrow x</td>
<td>true</td>
</tr>
<tr>
<td>2,3</td>
<td>Z\leftarrow x+y</td>
<td>true \land x \geq 3 \land x = x+3</td>
</tr>
<tr>
<td>5,6</td>
<td>Y\leftarrow y+5</td>
<td>x \geq 3 \land y &gt; 0</td>
</tr>
<tr>
<td>7,9</td>
<td>(x \geq 3 \land y &gt; 0 \land x-(y+5) \geq 0)</td>
<td>= (x \geq 3 \land y &gt; 0 \land (x-y) \geq 25)</td>
</tr>
</tbody>
</table>
Presenting the results

\begin{tabular}{|c|c|c|}
\hline
Statement & PV & PC \\
\hline
1 & $X \leftarrow x$ & true \\
2 & $Y \leftarrow y$ & \\
3 & $Z \leftarrow x+y$ & true; \(x \geq 3\) \\
4 & $Y \leftarrow y+5$ & \(x \geq 3\) \& \(y > 0\) \\
5 & $x \geq 3$ & \(y \leq 0\) \& \((x-y) < 0\) \\
6 & $x \geq 3$ & \(x \geq 3\) \& \(y \leq 0\) \& \((x-y) < 0\) \\
\hline
\end{tabular}

Results (feasible path)

\begin{align*}
P &= 1, 2, 3, 5, 6, 7, 9 \\
D[P] &= \{(x,y) | x \geq 3 \& y > 0 \& (x-y) \geq 5\} \\
C[P] &= PV.Y = y + 5
\end{align*}

Evaluating another path

\begin{tabular}{|c|c|c|}
\hline
Stmts & PV & PC \\
\hline
1 & $X \leftarrow x$ & true \\
2 & $Y \leftarrow y$ & \\
3 & $Z \leftarrow x+y$ & true; \(x \geq 3\) \\
4 & $Y \leftarrow y+5$ & \(x \geq 3\) \& \(y > 0\) \\
5 & $x \geq 3$ & \(y \leq 0\) \& \((x-y) < 0\) \\
6 & $x \geq 3$ & \(x \geq 3\) \& \(y \leq 0\) \& \((x-y) < 0\) \\
\hline
\end{tabular}

Results (infeasible path)

\begin{align*}
P &= 1, 2, 3, 5, 7, 8 \\
D[P] &= \{(x,y) | x \geq 3 \& y > 0 \& (x-y) < 0\} \\
C[P] &= PV.Y = y + 5
\end{align*}

Representing Computation

- Symbolic names represent the input values
- the path value PV of a variable for a path describes the value of that variable in terms of those symbolic names
- the computation of the path C[P] is described by the path values of the outputs for the path
Representing Conditionals
- An interpreted branch condition or interpreted predicate is represented as an inequality or equality condition.
- The path condition PC describes the domain of the path and is the conjunction of the interpreted branch conditions.
- The domain of the path D[P] is the set of input values that satisfy the PC for the path.

what about loops?
- Symbolic evaluation requires a complete path description.
- Example Paths:
  - P= 1, 2, 3, 5
  - P= 1, 2, 3, 4, 2, 3, 5
  - P= 1, 2, 3, 4, 2, 3, 5
  - Etc.

Symbolic Testing
- Path Computation provides a concise functional representation of behavior for the entire path.
- Examination of Path Domain and Computation often useful for detecting program errors.
- Particularly beneficial for (some) scientific applications or applications without oracles.

Simple Symbolic Evaluation
- Provides symbolic representations given path Pi:
  - Path condition: PC
  - Path domain: D[Pi] = \{(x1, x1, ..., x1) | pc true\}
  - Path values: PV.X1
  - Path computation: C[Pi] =

Additional Features:
- Simplification
- Path Condition Consistency
- Fault Detection
- Path Selection
- Test Data Generation

Simplification
- Reduces path condition to a canonical form.
- Simplifier often determines consistency:
  \[ PC = (x \geq 5) \land (x < 0) \]
- May want to display path computation in simplified and unsimplified form:
  \[ PV.X = x \cdot (x + 1) + (x + 2) + (x + 3) = 4 \cdot x + 6 \]
Path Condition Consistency

- strategy = solve a system of constraints
  - theorem prover
  - consistency
  - algebraic, e.g., linear programming
  - consistency and find solutions
  - solution is an example of automatically generated test data

... but, in general we cannot solve an arbitrary system of constraints!

Fault Detection

- Implicit fault conditions
  - E.g., Subscript value out of bounds
  - E.g., Division by zero e.g., Q=N/D
  - Create assertion to represent the fault and conjoin with the pc
  - Division by zero assert(divisor ≠ 0)
  - Determine consistency PC and (PV.divisor = 0)
    - if consistent then error possible
  - Must check the assertion at the point in the path where the construct occurs

Fault Detection (continued)

- Checking user-defined assertions
  - Determine if the complement of the assertion is consistent with the PC
    - example
      - Assert (A > B)
      - PC and (PV.A ≤ PV.B)
      - if consistent then assertion not valid

Comparing Dynamic and Symbolic Assertion Checking Approaches

- With run-time assertion checking, assertions are inserted as executable instructions and checked during execution
  - dependent on test data selected (dynamic testing)
- With symbolic evaluation, assertions checked when they occur on a path being evaluated
  - dependent on path, but not on the test data
  - look for violating data in the path domain

Additional Features:

- Simplification
- Path Condition Consistency
- Fault Detection
- Path Selection
- Test Data Generation

Path Selection

- User selected
- Automated selection to satisfy some criteria
  - e.g., exercise all statements at least once
- Because of infeasible paths, best if path selection done incrementally
**Incremental Path Selection**

- PC and PV maintained for partial path
- Inconsistent partial path can often be salvaged

\[ pc' = pc \text{ and } (x \leq 0) \]
\[ pc'' = pc' \text{ and } (x > 3) \]
\[ \text{INCONSISTENT!} \]
\[ \text{infeasible path} \]

\[ pc'' = pc' \text{ and } (x \leq 0) \]
\[ pc = pc' \text{ and } (x \leq 0) \]
\[ \text{CONSISTENT [if } pc' \text{ is consistent]} \]

**Path Selection (continued)**

- Can be used in conjunction with other static analysis techniques to determine path feasibility
- Testing criterion generates a path that needs to be tested
- Symbolic evaluation determines if the path is feasible
  - Can eliminate some paths from consideration

**Additional Features:**

- Simplification
- Path Condition Consistency
- Fault Detection
- Path Selection
- Test Data Generation

**Test Data Generation**

- Simple test date selection: Select test cases that satisfies the path condition pc
- Error based test date selection
  - Try to select test cases that will help reveal faults
  - Use information about the path domain and path values to select test data
    - e.g., \( PV.X = a \times (b + 2) \):
      - \( a = 1 \) combined with min and max values of \( b \)
      - \( b = -1 \) combined with min and max values for \( a \)
      - values that would force \( X \) to take on "special" values
        - e.g., \( a = 0 \) or \( b = 2 \), if \( X = 0 \) is a special value

**Enhanced Symbolic Evaluation Capabilities**

- Creates symbolic representations of the Path Domains and Computations
  - "Symbolic Testing"
- Determine if paths are feasible
- Automatic fault detection
  - system defined assertions
  - user assertions
- Automatic path selection
- Automatic test data generation
An Enhanced Symbolic Evaluation System

Symbolic Execution

Path Selection

Fault conditions

Symbolic Execution

Path condition

Simplifier

Inequality Solver

Fault report

Path condition

Domain test data

Path computation

Problems

- Information explosion
- Impracticality of all paths
- Path condition consistency
- Aliasing
  - elements of a compound type e.g., arrays and records
  - pointers

Alias Problem

Indeterminate subscript

Y := A(I)
Z := A(I)
I > 2
Y := A(I)
Z := A(I)

Escalating problem

- Read I
- X := A[I] PV.X = unknown
- Y := X + Z PV.Y = unknown + PV.Z = unknown

Can often determine array element

I := 0
I ≤ 3
Y := A(I)
I := I + 1

Symbolic Evaluation Approaches

- symbolic evaluation
  - With some enhancements
  - Data independent
  - Path dependent
- dynamic symbolic evaluation
  - Data dependent --> path dependent
- global symbolic evaluation
  - Data independent
  - Path independent
Dynamic Symbolic Execution

- Data dependent
- Provided information
  - Actual value:
    \[ X := 25.5 \]
  - Symbolic expression:
    \[ X := Y \times (A + 1.9); \]
  - Derived expression:
    \[ A \times 1.9 + (25.5) \]

Dynamic Analysis combined with Symbolic Execution

- Actual output values
- Symbolic representations for each path executed
  - path domain
  - path computation
- Fault detection
  - data dependent
  - path dependent (if accuracy is available)

Dynamic Symbolic Execution

- Advantages
  - No path condition consistency determination
  - No path selection problem
  - No aliasing problem (e.g., array subscripts)
- Disadvantages
  - Test data selection (path selection) left to user
  - Fault detection is often data dependent
    - Except for user and system defined assertions
- Applications
  - Debugging
  - Symbolic representations used to support path and data selection

Symbolic Evaluation Approaches

- simple symbolic evaluation
- dynamic symbolic evaluation
- global symbolic evaluation

- Data and path independent
- Loop analysis technique classifies paths that differ only by loop iterations
- Provides global symbolic representation for each class of paths

Global Symbolic Evaluation

- Loop Analysis
  - creates recurrence relations for variables and loop exit condition
  - solution is a closed form expression representing the loop
  - then, loop expression evaluated as a single node

Global Symbolic Evaluation

- Two classes of paths:
  - \( P_1: \{a, (1,2), 4, (5, (6, 7), 8), f\} \)
  - \( P_2: \{a, 3, 4, (5, (6, 7), 8), f\} \)

loop analysis
case
two cases:
endcase
- analyze the loops first
- consider all partial paths up to a node
Loop analysis example

read A, B
Area := 0
X := A

write AREA
AREA := AREA + A
X := X + 1

Loop Analysis Example

• Recurrence Relations
  \[ \text{AREA}_k = \text{AREA}_{k-1} + A_0 \]
  \[ X_k = X_{k-1} + 1 \]

• Loop Exit Condition
  \[ \text{lec}(k) = (X_k > B_0) \]

Loop Analysis Example (continued)

• solved recurrence relations
  \[ \text{AREA}(k) = \text{AREA}_0 + \sum_{i=0}^{k-1} A_i \]
  \[ X(k) = X_0 + k \]

• solved loop exit condition
  \[ \text{lec}(k) = (X_0 + k > B_0) \]

• loop expression
  \[ k_e = \min \{ k \mid X_0 + k > B_0 \text{ and } k \geq 0 \} \]
  \[ \text{AREA} = \text{AREA}_0 + \sum_{i=0}^{k_e} A_i \]
  \[ X = X_0 + k_e \]
  \[ \text{global representation for input } (a, b) \]
  \[ X_0 = a, A_0 = a, B_0 = b, \text{AREA}_0 = 0 \]
  \[ a + k_e > b \implies k_e \geq b - a \]
  \[ k_e = b - a + 1 \]
  \[ X = a + (b-a+1) = b+1 \]
  \[ \text{AREA} = \sum_{i=0}^{k_e} a = (b-a+1) a \]

Loop analysis example

read A, B
Area := 0
X := A

write AREA
AREA := AREA + A
X := X + 1

Find path computation and path domain for all classes of paths

• P1 = (1, 2, 3, 4, 7)
• D[P1] = a > b
• C[P1] = (\text{AREA} = 0) \text{ and } (X = a)
Find path computation and path domain for all classes of paths

- \( P_2 = (1, 2, 3, 4, (5, 6), 7) \)
- \( D[P_2] = (b \Rightarrow a) \)
- \( C[P_2] = \{ \text{AREA} = (b-a+1) a \} \)
- \( k_4 = b - a + 1 \)
- \( X = b + 1 \)

Example

```plaintext
procedure RECTANGLE (A,B: in real; H: in real range -1.0 ... 1.0; F: in array [0..2] of real; AREA: out real; ERROR: out boolean) is
    -- RECTANGLE approximates the area under the quadratic equation
    -- \( F[0] + F[1] \cdot X + F[2] \cdot X^2 \) from \( X=A \) to \( X=B \) in increments of \( H \).

    X, Y: real;

begin
    X := A;
    AREA := F[0] + F[1] \cdot X + F[2] \cdot X^2;
    while X + H \leq B loop
        X := X + H;
        Y := F[0] + F[1] \cdot X + F[2] \cdot X^2;
        AREA := AREA + Y;
    end loop;
    AREA := AREA \cdot H;
end RECTANGLE;
```

Symbolic Representation of Rectangle

```
X = b + 1
K = b - a + 1
A = a
B = b
\text{AREA} = (b-a+1) a
```

Global Symbolic Evaluation

- Advantages
  - global representation of routine
  - no path selection problem
- Disadvantages
  - has all problems of Symbolic Execution PLUS
    - inability to solve recurrence relations
      - (interdependencies, conditionals)
- Applications
  - has all applications of
    - Symbolic Execution plus
      - Verification
      - Program Optimization

Partial Evaluation

- Similar to Dynamic Symbolic Evaluation
- Provide some of the input values
  - E.g., If input is \( x \) and \( y \), provide a value for \( x \)
- Create a representation that incorporates those values and that is equivalent to the original representation if it were given the same values as the preset values
  - \( P(x', y) = P'(x', y) \)
**Why is partial evaluation useful?**

- In compilers
  - May create a faster representation
  - E.g., if you know the maximum size for a platform or domain, hardcode that into the system
- More than just constant propagation
  - Do symbolic manipulations with the computations

**Example with Ackermann's function**

- \( A(m,n) = \) if \( m = 0 \) then \( n+1 \) else
  - if \( n = 0 \) then \( A(m-1, 1) \) else
    - \( A(m-1, A(m,n-1)) \)
- \( A0(n) = n+1 \)
- \( A1(n) = \) if \( n = 0 \) then \( A0(1) \) else
  - \( A0(A1(n-1)) \)
- \( A2(n) = \) if \( n = 0 \) then \( A1(1) \) else
  - \( A1(A2(n-1)) \)
- ...

**Specialization using partial evaluation**

- \( A(2) := 5 \)
- \( \text{read } I, A(I) \)
- \( I > 2 \)
- \( Y := A(I) \)
- \( Z := A(2) \)

- \( I = 2 \)
- \( Z := \text{eval}(A(2)) \)

**Why is Partial Evaluation Useful in Analysis**

- Often can not reason about dynamic information
  - Instantiates a particular configuration of the system that is easier to reason about
  - E.g., the number of tasks in a concurrent system; the maximum size of a vector
- Look at several configurations and try to generalize results
  - Induction
  - Often done informally

**Reference on Partial Evaluation**

**Symbolic Evaluation Approaches**

- Symbolic evaluation
  - With some enhancements
  - Data independent
  - Path dependent
- Dynamic symbolic evaluation
  - Data dependent --> path dependent
- Global symbolic evaluation
  - Data independent
  - Path independent

**Current Status of Symbolic Execution**

- Not generally used for symbolic testing
  - Expensive to create representations
  - Expensive to reason about expressions
  - Imprecision of results
- But, being incorporated into other techniques
  - Test data generation with assertions
  - Eliminate infeasible counter examples
- Concolic testing
  - JCUTE for Java
- Current computing power and better user interface capabilities make it worth considering